

Oktáv továbbképző központ

Szakképesítés neve: Szoftverfejlesztő

Képzés OKJ száma: 54 213 05

Szakdolgozat

Car rental manager

Témavezető:

Blahut Lóránt

Készítette:

Györke Iván Mátyás

Szoftverfejlesztő

Budapest

2022

1 TARTALOMJEGYZÉK

2 Bevezetés.....	4
3 Felhasználói dokumentáció	4
3.1 A program célja.....	4
3.2 Rendszerkövetelmények.....	5
3.2.1 Windows.....	5
3.2.2 Linux.....	5
3.2.3 Szoftverek	5
3.3 A program leírása	6
3.3.1 A bejelentkező ablak:	6
3.3.2 Autók ablak.....	7
3.3.3 Autó bérlese ablak.....	9
3.3.4 Autó visszavétele ablak.....	11
3.3.5 Ügyfelek ablak.....	12
3.3.6 Kijelentkezés, bezárás	14
4 Fejlesztői dokumentáció	15
4.1 Adatbázis	15
4.1.1 rental_users tábla.....	15
4.1.2 rental_rents tábla.....	16
4.1.3 rental_returns tábla	16
4.1.4 rental_cars tábla.....	16
4.1.5 rental_admins tábla	17
4.2 Dependenciák	17
4.3 Osztályok	17
4.3.1 Db osztály.....	18
4.3.2 Splash osztály	18
4.3.3 Login osztály	19
4.3.4 Cars osztály.....	20
4.3.5 NewCar osztály	21
4.3.6 EditDeleteCar osztály	22
4.3.7 Customers osztály	24

4.3.8	NewCustomer osztály	26
4.3.9	EditDeleteCustomer osztály	26
4.3.10	CarRent osztály.....	28
4.3.11	CarReturn osztály	32
5	Összefoglalás	40
5.1	Szakdolgozatom célja.....	40
5.2	Megvalósítás.....	41
5.3	Fejlesztési lehetőségek.....	41
6	Összefoglalás	Hiba! A könyvjelző nem létezik.

2 BEVEZETÉS

Szoftverfejlesztő szakdolgozatom témája a **Car rental manager** nevű Java programozási nyelven megvalósított ablakos alkalmazás, amely egy autók bérbeadásával foglalkozó cég belső rendszerét hivatott megvalósítani. A program használata egyszerű, ezért egy autó bérbeadó cég bármely alkalmazottja által rövid időn belül elsajátítható.

Azért erre a rendszerre esett a választásom, mivel érdekelnek a vállalatirányítási rendszerek, és ennek a programnak az elkészítésével magam is szélesebb körű betekintést nyerhettem ezen rendszerek elkészítésének folyamatába.

A program JAVA programozási nyelven készült, az **Eclipse IDE** programban. A Felületet a JAVA **WindowBuilder** segítségével hoztam létre, a funkciókat pedig objektum orientált szemléletben készítettem el hozzá, igyekezve betartani a tiszta kód szabályait. Az adatbázist **mySQL** technológiát használtam, és a **mySQL workbench** programban készítettem el.

3 FELHASZNÁLÓI DOKUMENTÁCIÓ

3.1 A PROGRAM CÉLJA

A program egy autó bérbeadó cég belső rendszerét hivatott megvalósítani, mely képes kezelní a cégen belül az ügyfelek, az autók, és az éppen kibérelt és már visszahozott autók nyilvántartását, és az adott bérletek árait.

3.2 RENDSZERKÖVETELMÉNYEK

Mivel a program Java 8-ban készült, így a rendszerkövetelményei a következők:

3.2.1 Windows

- Windows 10 (8u51 and above)
- Windows 8.x (Desktop)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64-bit)
- Windows Server 2012 and 2012 R2 (64-bit)
- RAM: 128 MB
- Disk space: 124 MB for JRE; 2 MB for Java Update
- Processor: Minimum Pentium 2 266 MHz

3.2.2 Linux

- Oracle Linux 5.5+¹
- Oracle Linux 6.x (32-bit), 6.x (64-bit)²
- Oracle Linux 7.x (64-bit)² (8u20 and above)
- Red Hat Enterprise Linux 5.5+¹ 6.x (32-bit), 6.x (64-bit)²
- Red Hat Enterprise Linux 7.x (64-bit)² (8u20 and above)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64-bit)² (8u31 and above)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 and above)
- Ubuntu Linux 15.04 (8u45 and above)
- Ubuntu Linux 15.10 (8u65 and above)

3.2.3 Szoftverek

- a mySQL szerver futtatása jelen esetben parancs sorból történik, melyhez a **mysql-8.0.20-winx64** verzióját használtam.
- Az adatbázis kezeléshez a **mySQL Workbench** volt segítségemre.
- A program fejlesztéséhez és futtatásához az **Eclipse IDE** felületét használtam.

3.3 A PROGRAM FUTTATÁSA

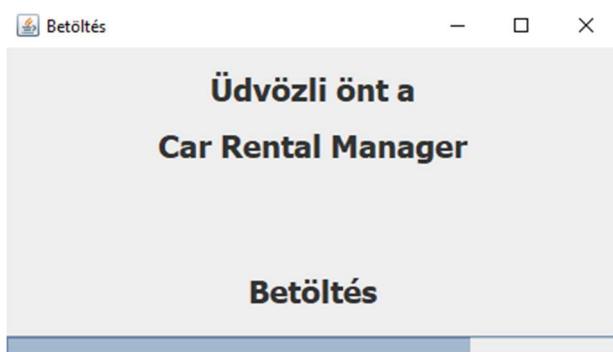
Jelenlegi állapotában a program az Eclipse IDE programmal futtatható, és az adatbázis lokális létrehozását igényli. Erre a fejlesztői dokumentációban térek ki

bővebben. A lokális szerver beállításai a DB osztályban szabadon módosíthatók, alap beállításon a 13306. portra kell létrehozni az adatbázist.

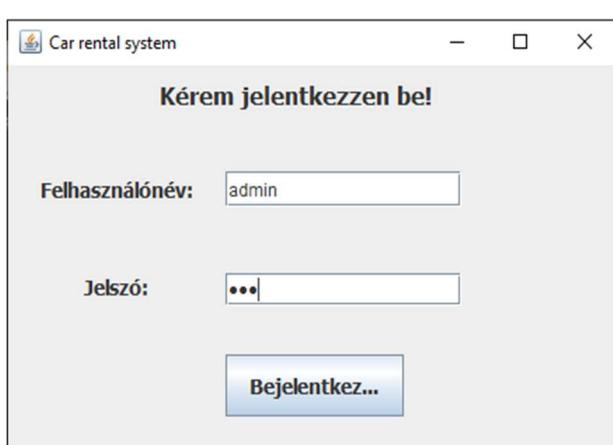
3.4 A PROGRAM LEÍRÁSA

Első lépéskét elindítjuk a lokális mySQL szervert. Ez gyakorlati alkalmazásban ki kell kerüljön egy webszerverre, amely 0/24-ben fut, jelen alkalmazásban ezt manuálisan kell megtenni minden indításnál, erre a fejlesztői dokumentációban térek ki.

A mySQL szerver indítása után a program készen áll a használatra. A program indításakor egy üdvözlő és betöltő képernyő jelenik meg, majd miután betöltött, egy bejelentkező ablakban kell bejelentkezni, ezzel is meggyártva a rendszer jogosulatlan elérését.



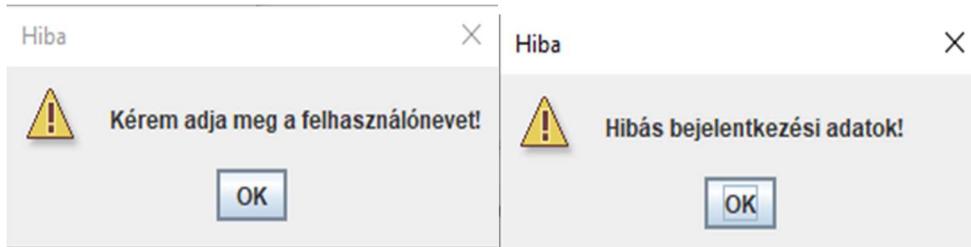
3.4.1 A bejelentkező ablak:



Jelen esetben a következő bejelentkezási adatok vannak beállítva a mySQL szerveren:

felhasználónév: admin
jelszó: pwd

Hibás, vagy hiányzó bejelentkezási adatok esetén a rendszer hibaüzenetet dob:



3.4.2 Autók ablak

Bejelentkezés után az autók ablak jelenik meg, ahol a cégnél lévő összes autót láthatjuk, újakat vihetünk fel, valamint a meglévők bizonyos adatait módosíthatjuk, például egy esetleges rendszám, vagy napi ár változást meg tudunk rajta valósítani. Ezen kívül innen tudunk tovább navigálni az autók bérletéhez, autók visszavételéhez, és az ügyfelek adatainak kezeléséhez.

A felület a következőképpen néz ki:

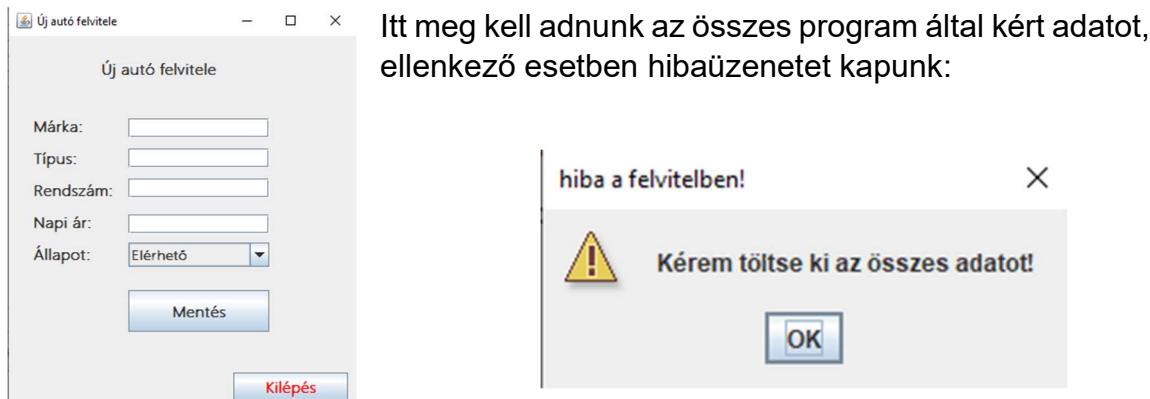
ID	Márka	Típus	Rendszám	Státusz	Ár
18	Lada	Samara	GAS-31	elérhető	300000
19	suzuki	swift	KSL-123	foglalt	5560
20	suzuki	alto	RLD-315	foglalt	42342
21	skoda	fabia	412-312	elérhető	30000
22	Mazda	3	SLD-323	foglalt	8000
23	Ferrari	423	PENZ-30	foglalt	200000
24	Lamborghini	Urus	PTL-361	foglalt	300000

Buttons at the bottom:

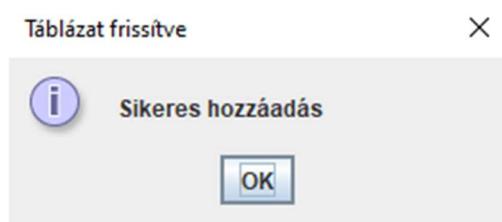
- Autó bérlese
- Autó visszavétele
- Ügyfelek megjelenítése
- Új autó felvitele
- Kijelentkezés (highlighted in red)

3.4.2.1 Új autó felvitele

Ha rákattintunk az Új autó felvitele gombra, előugrik egy ablak ahol megadhatjuk a felvinni kívánt autó adatait.



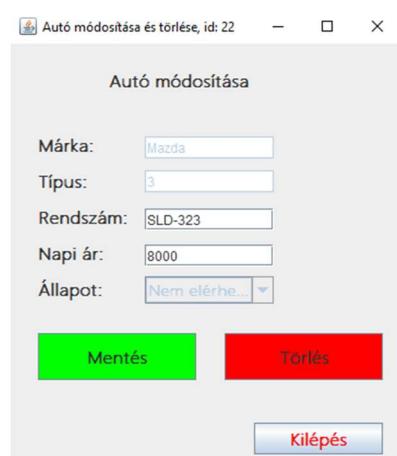
Ha az autót sikeresen felvittük, a program erről is egy értesítést dob fel:



3.4.2.2 Autó módosítása/törlése ablak

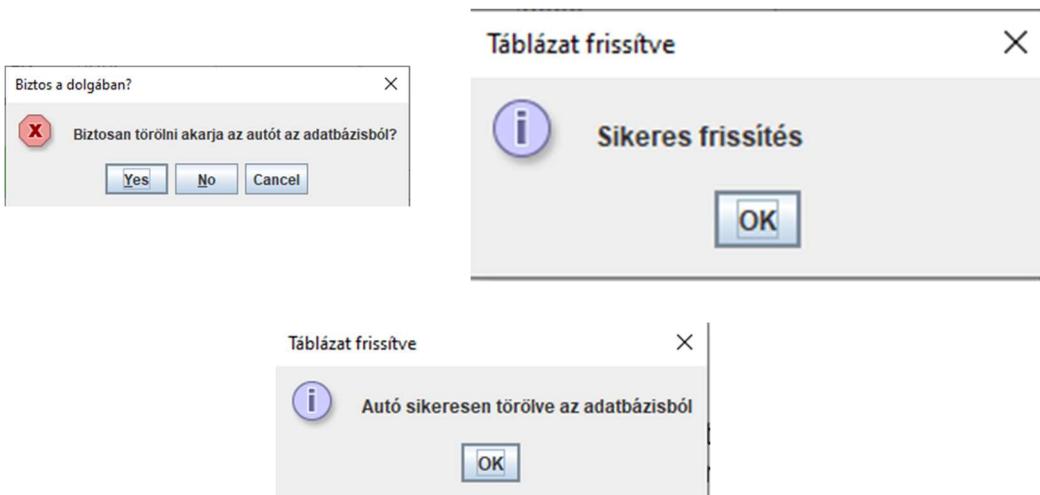
Ha az autók ablakon lévő táblázaton rákattintunk egy autóra, megjelenik az autó módosítása/törlése ablak, felül az autó azonosítójával, ahol módosíthatjuk az autók rendszámát, illetve napi árát módosíthatjuk. A program készítésénél feltételeztem hogy egy autó márka és típusa nem változhat, így ezek nem módosíthatók. Az autó státusza a bérbeadással és visszavételével módosítható.

Az autó adatainak módosításához a mentés gombra kell kattintanunk, az adatbázisból való törléséhez pedig a törlés gombra. Mentés gombra kattintva a



program egy felugró ablakban értesít minket:

Törlés gombra kattintva pedig rá kérdez hogy biztos törölni akarjuk e az autót, majd ezután értesít egy felugró ablakban ha igennel válaszoltunk:



3.4.3 Autó bérlese ablak

Az autó bérlese gombra kattintva jelenik meg az autó foglalás ablak, ahol új bérléseket vihetünk fel a rendszerbe. Ezen az ablakon van egy szabad és egy foglalt autók táblázat. A szabad autók egyikére kattintva automatikusan kitölți a rendszám, márka, típus, valamint a bérlet kezdete mezőket, utóbbit minden az aznapi dátummal.

Autó foglalás

Autó foglalás

Szabad autók					
ID	Márka	Típus	Rendszám	Státusz	Ár
18	Lada	Samara	GAS-31	elérhető	300000
19	suzuki	swift	KSL-123	elérhető	5560
20	suzuki	alto	RLD-315	elérhető	42342
23	Ferrari	423	PENZ-30	elérhető	200000
24	Lamborghini	Urus	PTL-361	elérhető	300000
26	KIA	Ceed	RSD-312	elérhető	9000

Foglalt autók								
Bérítés ID	Autó ID	Bérítő neve	Márka	Típus	Rendszám	Bérítés kezdete	Bérítés vége	Teljes ár
10	22	Teszt Elek	Mazda	3	SLD-323	2022-05-07	2022-05-11	32000
15	21	Teszt Elek	skoda	fabia	412-312	2022-05-10	2022-05-15	150000

Rendszám	Márka	Típus	Ár
PENZ-30	Ferrari	423	800000
Bérítés kezdete		Bérítés vége	
2022.05.10.	<input type="button" value="Vissza"/>	2022.05.14.	<input type="button" value="Vissza"/>
Ügyfél			
Teszt Elek		<input type="button" value="Mentés"/>	
<input type="button" value="Nyomtatás"/>		<input type="button" value="Alaphelyzet"/>	
		<input style="background-color: red; color: white; font-weight: bold;" type="button" value="Kilépés"/>	

Fontos, hogy a bérítés mindenkorábban a rendszerbe felvitel napjától indul, a visszahozatal dátumát pedig manuálisan kell megadni.

Ezután a számítás gombra kell kattintani, ahol a program kiszámolja a bérelt napokat, és ebből a teljes árat.

Ezután már nincs más dolgunk, mint kiválasztani az ügyfelet egy legördülő fülön, majd a Mentés gombra kattintva ezt rögzíteni a rendszerben. Ezt követően a program a főoldalon frissíti az autó státuszát foglaltra, valamint a bérítések ablakban rögzíti a kibérelt autók listába, a szabadakból pedig kikerül, ezzel is gátolva a már kiadott autók esetleges hibás újra kibérlését.

Az ablakon helyet kapott még egy alaphelyzet gomb is, ha esetleg több autót szeretnénk egymás után bérbe adni. Ez értelemszerűen alaphelyzetbe állítja az összes bevíteli mezőt, ezt követően kiválaszthatunk egy új autót.

A Nyomtatás gombra kattintva előugrik egy Windows nyomtatási ablak, ahol kinyomtathatjuk az éppen kiadott autók listáját. Ez történhet pdf formátumban a számítógépre is, illetve természetesen papír formában nyomtatóra küldve.

A Kilépés gombbal térhetünk vissza a fő ablakra.

3.4.4 Autó visszavétele ablak

Az **autó visszavétele** gombra kattintva megnyílik az autó visszavét ablak, ahol a bérítés ablakhoz hasonló módon két táblázattal találkozunk. Az első tábla az éppen kibérelt autók, a második, alsó tábla pedig a már visszahozott autók.

Használatában is hasonlóan működik, mint az autó bérítése ablak. Itt a kibérelt autók táblázatból kell kiválasztanunk egy autót. A kiválasztott autó adatai információi kiírásra kerülnek a szöveg mezőkben, módosítani ezeket nem lehet.

The screenshot shows a software window titled 'Bérelt autó visszaszolgáltatása'. The main title is 'Autó visszavétele'. On the left, there are several search filters: 'Rendszám' (SLD-323), 'Márka' (Mazda), 'Típus' (3), and 'Ügyfél neve' (Teszt Elek). Below these is a section for 'Visszahozás dátuma' (2022.05.12.) and 'Késés' (1). There is also a 'Felár' field (8000) with a 'Kiszámítás' button next to it. At the bottom left is a 'Visszavétel' button. In the center, there are two tables. The top table, 'Kibérelt autók', contains the following data:

Bérítés ID	Autó ID	Bérítő neve	Márka	Típus	Rendszám	Bérítés kez.	Bérítés vége	Teljes ár
10	22	Teszt Elek	Mazda	3	SLD-323	2022-05-07	2022-05-11	32000
15	21	Teszt Elek	skoda	fabia	412-312	2022-05-10	2022-05-15	150000

The bottom table, 'Visszahozott autók', contains the following data:

ID	Rendszám	Bérítő neve	Visszahozás dátuma	Ár
34	PENZ-30	Polgár Janika	2022-05-08	0
35	324.3123	Teszt Elek	2022-05-08	0
36	PENZ-30	Teszt Péter	2022-05-08	200000
37	SLD-323	Polgár Jenő	2022-05-09	8000
38	324.3123	Teszt Elek	2022-05-10	42342
39	412-312	Teszt Elek	2022-05-10	60000
40	SLD-323	Polgár Jenőke	2022-05-10	16000
41	PENZ-30	Teszt Péter	2022-05-11	0
42	324.3123	Teszt Elek	2022-05-14	296394
43	Klk-123	Teszt Elek	2022-05-13	22240
44	Klk-123	Teszt Elek	2022-05-13	22240
45	303030	Teszt Elek	2022-05-15	440000
46	PTL-361	Polgár Jenő	2022-05-13	900000

At the bottom right are 'Nyomtatás' and 'Kilépés' buttons.

Az autó kiválasztása után meg kell adnunk azt a dátumot amikor visszahozták az autót, majd ezt összevetve a táblázatban látható „bérítés vége” dátummal összevetve, manuálisan meg kell adnunk hány napot késsett vele a bérítő, amennyiben késsett. Amennyiben időben hozta vissza az autót ide 0 értéket írunk be.

Ezután a **Kiszámítás** gombra kattintunk, amely a késés felárát számolja ki.

Ha kiszámoltuk az esetleges késés felárát, már nincs más dolgunk, mint a **Visszavétel** gombra kattintani, amely rögzíti az autó visszahozatalát, és

kiszámolja a teljes árat az előzetesen kiszámolt árból, valamint hozzáadja az esetleges késés felárát. **Így a visszahozott autók táblában már a bérlet teljes ára fog látszani.**

Itt is helyet kapott egy **Nyomtatás** gomb, amellyel pdf formátumban menthetjük, illetve ki is nyomtathatjuk a visszahozott autók listáját.

A **Kilépés** gombbal visszatérhetünk az autók ablakhoz.

3.4.5 Ügyfelek ablak

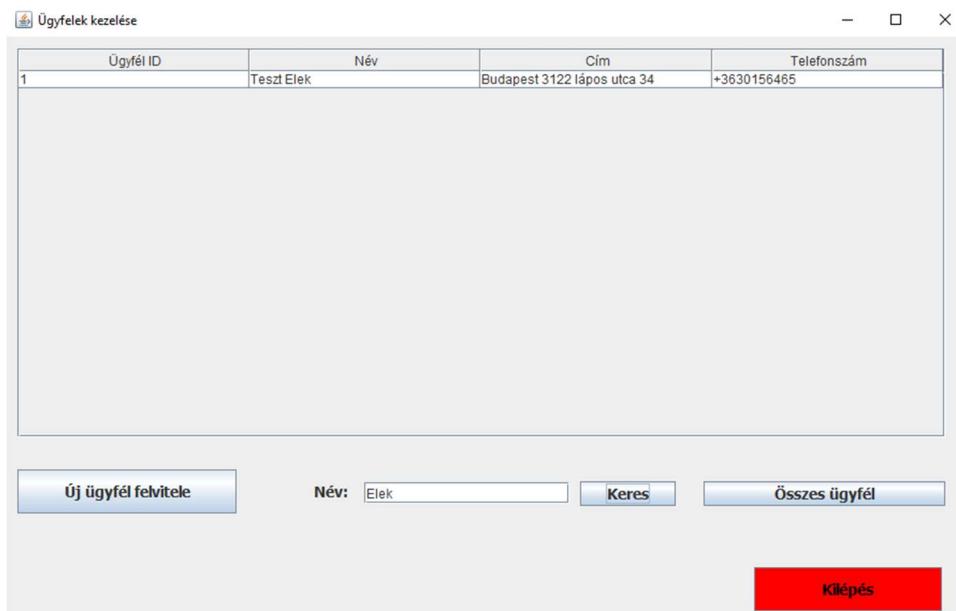
Az **Ügyfelek** gombra kattintva nyithatjuk meg az ügyfelek adminisztrálására szolgáló ablakot. Itt helyet kap egy táblázat, amelyben alapértelmezetten az összes ügyfél adatait láthatjuk.

Ügyfél ID	Név	Cím	Telefonszám
1	Teszt Elek	Budapest 3122 Lápos utca 34	+3630156465
2	Polgár Jenő úr	Debrecen 1562 Tégláégető utca 34	+36507829452
3	Polgár Janika	Debrecen 1562 Tégláégető utca 34	+36507829452
4	Polgár Jenőke	Budapest 1204 fő utca 1	+36508792451
6	Polgár Jenő	Debrecen 1562 Tégláégető utca 34	+36507829452
9	Takács Jánai	Budapest 1230 Mocsaras utca 34	+36507829452
10	Polgár Jenő úr	Debrecen 1562 Tégláégető utca 34	06304579751

Új ügyfél felvitele Név: Keres Összes ügyfél

Kilépés

Az ügyfelek szűrésére belekerült egy keresés funkció, amellyel az ügyfelek nevére szűrhetünk rá. Itt a be kell írni a keresett nevet vagy név részletet, majd a keresés gombra kattintani. Ezután a táblázatban már csak azok a személyek fognak látszani, akiknek a nevében szerepel a keresett részlet.



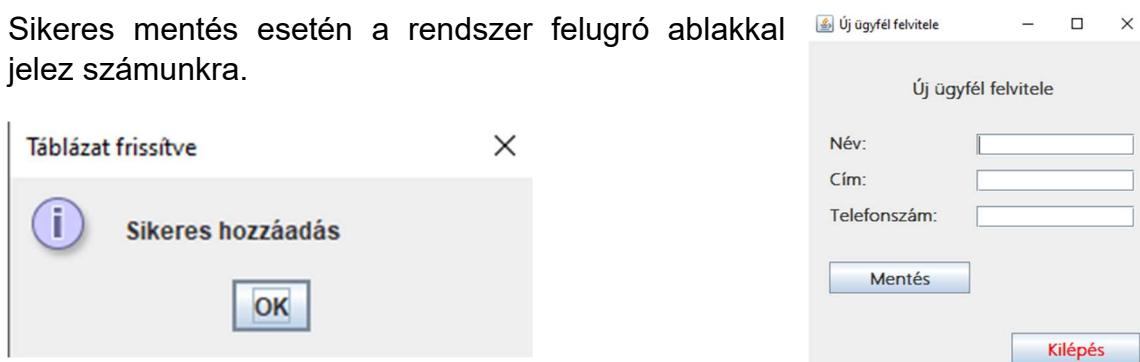
Amennyiben keresés után újra a teljes táblázatot látni szeretnénk, nincs más dolgunk, mint az összes ügyfél gombra kattintani.

3.4.5.1 Új ügyfél felvitele

Amennyiben új ügyfelet szeretnénk felvinni a rendszerbe, az **Új ügyfél felvitele** gombra kattintva hozhatjuk elő az erre szolgáló ablakot.

A név, cím, telefonszám megadása után a **Mentés** gombra kattintva rögzíthetjük a rendszerben az ügyfelet, akit ezután már hozzá lehet rendelni a bérlekhez.

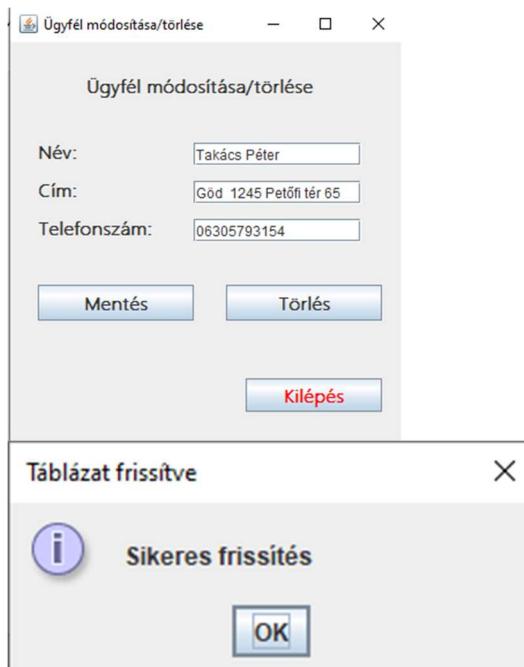
Sikerességek esetén a rendszer felugró ablakkal jelez számunkra.



3.4.5.2 Ügyfél módosítása/törlése

Az ügyfél módosítása/törlése ablakot hasonló módon hozhatjuk elő, mint az autó módosítása/törlése ablakot, csak rá kell kattintanunk egy ügyfélre a táblázatban.

A **Mentés** gombra kattintva itt is felugró ablak jelzi a sikeres mentést.



A **Törlés** gombra kattintva itt is először rágérdez a program, hogy biztos törölni szeretnénk e az ügyfelet, majd ha igen a válasz akkor felugró ablakkal jelzi ennek sikerességét.



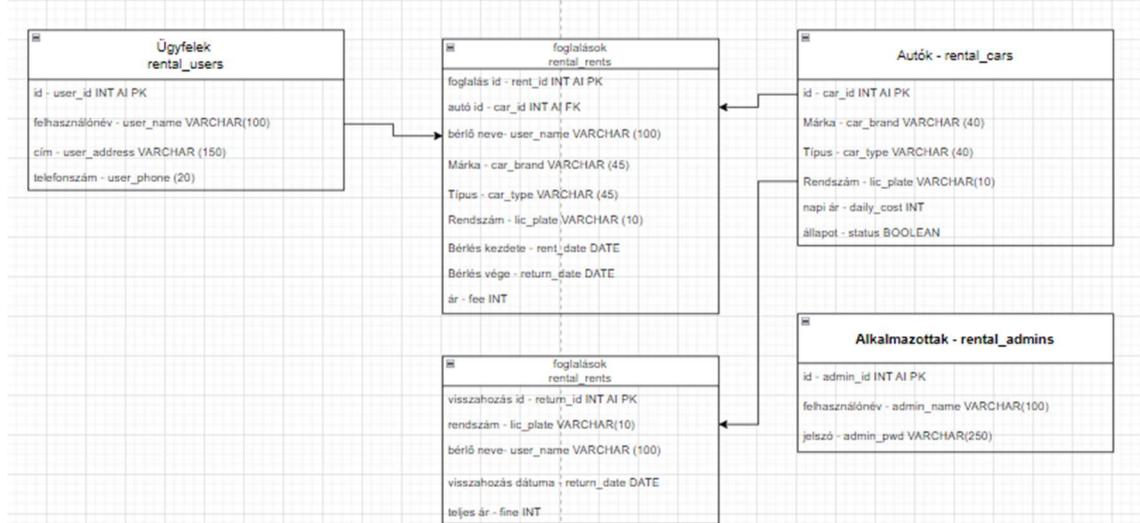
3.4.6 Kijelentkezés, bezárás

Ha végeztünk a program használatával, az autók ablakon található **Kijelentkezés** gombbal térhetünk vissza a bejelentkező felületre, ezt bezárva pedig bezárhatjuk magát a programot is. A program bezárható még az **Autók** ablak bezárásával.

4 FEJLESZTŐI DOKUMENTÁCIÓ

4.1 ADATBÁZIS

Az adatok tárolására egy **MySQL** adatbázist hoztam létre, amelyet a **MySQL workbench** segítségével készítettem. Jelen esetben az adatbázis egy lokális



webszerverre lett telepítve, amely a 13306. porton található. Ez a **DB** osztály módosításával bármikor megváltoztatható, az adatbázis telepítő állományból bármely más adatbázisba feltelepítve és a **DB** osztályon belül az elérési utat megváltoztatva elérhető.

Az adatbázis telepítő fájljait a mellékletben lehet megtalálni.

Táblára lehet bontani az adatbázist, ezek szolgálnak az ügyfelek, a futó és lezárt bérlesek, az autók, valamint az alkalmazottak adatainak tárolására.

4.1.1 rental_users tábla

Az ügyfelek adatainak tárolására a **rental_users** tábla szolgál, az **user_id** az ügyfelek egyedi azonosítója, auto increment beállítással. Az **user_name** az ügyfelek nevének tárolására szolgál. Az **user_address** a címük tárolására, az **user_phone** pedig a telefonszámuknak.

Ügyfelek
id - user_id INT AI PK
felhasználónév - user_name VARCHAR(100)
cím - user_address VARCHAR (150)
telefonszám - user_phone (20)

4.1.2 rental_rents tábla

Az éppen futó bérlesek tárolására a **rental_rents** tábla nyújt megoldást. A **rent_id** ezek egyedi azonosítója, auto increment beállítással, ez a primary key is a táblában. A **car_id** idegen kulcs, ami a **rental_cars** táblából érkezik. Az **user_name** a bérő neve, ami a programból kerül feltöltésre, csakúgy mint a **car_brand**, **car_type**, **lic_plate**, **rent_date**, **return_date**, **fee** elemek is.

A **fee** a bérlet megkezdésekor előre kiszámolt összeg, ez a visszahozásnál még nőhet a későbbi visszahozás tekintetében.

foglalások rental_rents	
foglalás id -	rent_id INT AI PK
autó id -	car_id INT AI FK
bérő neve -	user_name VARCHAR (100)
Márka -	car_brand VARCHAR (45)
Típus -	car_type VARCHAR (45)
Rendszám -	lic_plate VARCHAR (10)
Bérlet kezdete -	rent_date DATE
Bérlet vége -	return_date DATE
ár -	fee INT

4.1.3 rental_returns tábla

A befejezett bérleseknek is külön táblát hoztam létre, ebben szerepel a **return_id**, ami ezeknek az egyedi azonosítója auto increment beállítással, valamint a rendszám – **lic_plate**, ami a programból kerül fel, szint-úgy mint a bérő neve – **user_name**, a visszahozás dátuma – **return_date**, és a teljes ár – **fine**.

visszahozott autók rental_returns	
visszahozás id -	return_id INT AI PK
rendszer -	lic_plate VARCHAR(10)
bérő neve -	user_name VARCHAR (100)
visszahozás dátuma -	return_date DATE
teljes ár -	fine INT

4.1.4 rental_cars tábla

Az autók nyilvántartására a **rental_cars** tábla szolgál. Ebben helyet kap az autók egyedi azonosítója a **car_id** auto incrementre beállítva, a márka – **car_brand**, az autó típusa – **car_type**, a rendszám – **lic_plate**, a napi ár – **daily_cost**, és az állapot – **status** ami azt jelöli hogy az autó éppen foglalt vagy szabad-e.

Autók - rental_cars	
id -	car_id INT AI PK
Márka -	car_brand VARCHAR (40)
Típus -	car_type VARCHAR (40)
Rendszám -	lic_plate VARCHAR(10)
napi ár -	daily_cost INT
állapot -	status BOOLEAN

4.1.5 rental_admins tábla

Az alkalmazottak nyilvántartására a **rental_admins** tábla nyújt megoldást.

Az **admin_id** az alkalmazottak egyedi azonosítója, ami auto increment-re van állítva. A felhasználónév az **admin_name** mezőben kap helyet, a jelszó pedig az **admin_pwd** mezőben.

Alkalmazottak - rental_admins	
id - admin_id	INT AI PK
felhasználónév - admin_name	VARCHAR(100)
jelszó - admin_pwd	VARCHAR(250)

4.2 DEPENDENCIÁK

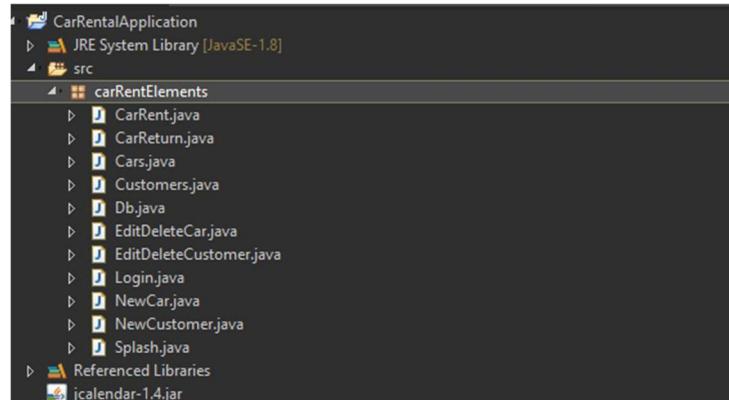
A program futtatásához elengedhetetlenek a következő JAR file-ok:

- mysql-connector-java-5.1.46.jar
- jcalendar-1.4.jar
- JRE system library [JavaSE-1.8]

4.3 OSZTÁLYOK

A program osztályai valósítják meg a különböző ablakokat, az ezekben található metódusok teszik lehetővé a program működését.

Az osztályok a carRentElements package-n belül találhatóak.



4.3.1 Db osztály

A Db osztály felelős az adatbázis kapcsolatért. Jelen esetben a localhost 13306. portja van beállítva, az adatbázis neve pedig car_rent, ami egyúttal a felhasználónév és a jelszó is az adatbázishoz. Ez később szabadon változtatható, ha esetleg online szerverrel szeretnénk használni a programot.

Metódusai:

```
public class Db {
    private Connection con;
    public Db() {
        try {
            con = DriverManager.getConnection(
                "jdbc:mysql://localhost:13306/car_rent?autoReconnect=true&useSSL=false",
                "car_rent",
                "car_rent"
            );
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public Connection getCon() {
        return con;
    }

    public ResultSet lekerdez(String sql) {
        ResultSet rs = null;
        Statement stm = null;
        try {
            stm = con.createStatement();
            rs = stm.executeQuery(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return rs;
    }
}
```

- **getCon()** – visszatér a **Connection** típusú **con** változóval, amelynek megadtuk a csatlakozási adatokat. Ezt felhasználhatjuk a többi osztálynál, ha adatbázis műveleteket szeretnénk végezni
- **lekerdez(String sql)** – Egy lekérdezésre használható metódus, egy String típusú sql utasítást vár, és egy **ResultSet** típusú **rs** változóval tér vissza.

4.3.2 Splash osztály

A Splash osztály a program belépési pontja. Egyedül a Main metódus kap benne helyet, ezen belül pedig van egy függvény ami a betöltő képernyő alján lévő **sp1**

progressBar betöltéséért felel, kivételkezeléssel megoldva. Miután ez lefutott a bejelentkező ablakra irányít, magát pedig bezárja.

```
@SuppressWarnings("serial")
public class Splash extends JFrame {

    private JPanel contentPane;
    private JProgressBar progBar1;

    public static void main(String[] args) {
        Splash sp1 = new Splash();
        sp1.setVisible(true);

        try {
            for (int i = 0; i < 100; i++) {
                Thread.sleep(20);
                sp1.progBar1.setValue(i);
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Hiba a betöltésben", "Hiba", JOptionPane.ERROR_MESSAGE);
        }
        new Login();
        sp1.dispose();
    }
}
```

4.3.3 Login osztály

A Login osztály felel a bejelentkezésért.

```
private JFrame frmLogin;
private JTextField tfUserName;
private JLabel lblMain;
private JLabel lblUserName;
private JLabel lblPwd;
private JButton btnBejelentkezes;
private JPasswordField pfPwd;
```

Elemei:

Metódusai:

- **carsAblakMegjelenít()** – A helyes bejelentkezási adatok megadása után megjeleníti a **Cars** ablakot, és eltűnik. Hiányzó felhasználónévnél vagy jelszónál, illetve hibás bejelentkezási adatok megadásakor is hiba ablakkal tér vissza. A bejelentkezási adatok az adatbázis rental_admins táblájában vannak tárolva, innen **PreparedStatement** segítségével vannak lekérdezve az adatok. Ez a metódus a bejelentkezés gomb (**btnBejelentkezes**) megnyomására lesz meghívva.

```
String sql = "SELECT * FROM rental_admins WHERE admin_name=? AND admin_pwd=?;"
```

```
PreparedStatement ps;
try {
    ps = con.prepareStatement(sql);
    ps.setString(1, userName);
    ps.setString(2, password);
    ResultSet rs = ps.executeQuery();
    if(rs.next()) {
        frmLogin.setVisible(false);
        dispose();
        new Cars();
    }
}
```

4.3.4 Cars osztály

Az alkalmazás legfontosabb osztálya, ez szolgál úgymond kezdőlapként, innen érhető el a többi ablak, és itt láthatók az autók a **tblAutok** táblázatban. Fontos, hogy az autók státusza itt szövegesen jelenik meg a táblázatban, ez a **tablazatBetolt** metódusban van megoldva.

Elemei:

```
private JFrame frmCarRentalManager;
private JTable tblAutok;
private JScrollPane scrollPane;
private JButton btnUjAuto;
private JButton btnKilps;
private JButton btnBerles;
private JButton btnUgyfelek;
private JButton btnVissza;
```

Metódusai:

```
public void tablazatBetolt() {}

public void tablazatSorokTorol() {}

public void modositEsTorolAblakMegjelenit(int tableRowIndex) {}

public void UjAutoAblakMegjelenit() {}

public void ugyfelekAblakMegjelenit() {}

public void ujBerlesAblakMegjelenit() {}

public void autoVisszaAblakMegjelenit() {}
```

A **tablazatBetolt()** funkció felel az autók tábla betöltéséért. Meghívásakor törli a táblázat tartalmát és lekérdezi az összes autót a **rental_cars** táblából, id szerint sorba rendezi őket, az autó státuszát pedig szövegesen írja ki a táblázatba.

```
public void tablazatBetolt() {
    Db dbObj = new Db();
    String sql = "SELECT * FROM rental_cars ORDER BY car_id";
    ResultSet rs = dbObj.lekerdez(sql);
    tablazatSorokTorol();
    try {
        while(rs.next()) {
            String allapot = rs.getBoolean("status") ? "elérhető" : "foglalt";
            ((DefaultTableModel)tblAutok.getModel()).addRow(
                new Object[] {
                    rs.getString("car_id"),
                    rs.getString("car_brand"),
                    rs.getString("car_type"),
                    rs.getString("lic_plate"),
                    allapot,
                    rs.getString("daily_cost"),
                });
        }
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

tablazatSorokTorol() funkció felelős a táblázat kiürítéséért, ennek a táblázat frissítésekor van fontos szerepe.

```
public void tablazatSorokTorol() {
    int sorokSzama = tblAutok.getModel().getRowCount();
    for (int i = sorokSzama-1; i >= 0; i--) {
        ((DefaultTableModel)tblAutok.getModel()).removeRow(i);
    }
}
```

modositEsTorolAblakMegjelenit(int rowIndex): Ez a funkció jeleníti meg egy kiválasztott autónál a módosítás/törlés ablakot. A rowIndex automatikusan megkapja mikor egy autóra kattintunk a táblázaton, majd ezalapján megkeresi az autó id-ját, és ezt továbbadja az ablaknak.

```
public void modositEsTorolAblakMegjelenit(int rowIndex) {
    int id = Integer.parseInt(((DefaultTableModel)tblAutok.getModel()).getValueAt(rowIndex, 0).toString());
    new EditDeleteCar(this, id);
}
```

ujAutoAblakMegjelenit(): Megjeleníti a NewCar ablakot.

ugyfelekAblakMegjelenit(): Megjeleníti az Customers ablakot.

ujBerlesAblakMegjelenit(): Megjeleníti a CarRent ablakot

autoVisszaAblakMegjelenit(): Megjeleníti a CarReturn ablakot.

4.3.5 NewCar osztály

Az új autók felvitelére szolgáló ablak, a **Cars** ablak **btnUjAuto** gombjára kattintva hívhatjuk meg.

Elemei:

```
private JPanel cpUjAuto;
private JTextField tfRendszam;
private JTextField tfTipus;
private JTextField tfMarka;
private JTextField tfNapiAr;
@SuppressWarnings("rawtypes")
private JComboBox cbAllapot;
private JButton btnMentes;
private JButton btnKilepes;
private JLabel lblUjAuto;
private JLabel lblMarka;
private JLabel lblTipus;
private JLabel lblRendszam;
private JLabel lblNapiAr;
private JLabel lblAllapot;
```

Metódusai:

```
private void MentesGombraKattintas(Cars MainWindow) {
```

A **btnMentes** gombra kattintva hívjuk meg ez a metódus, amely először ellenőrzi hogy minden szövegmező ki van e töltve, ezek hiányában hibaüzenetet dob.

```
if(tfMarka.getText().isEmpty() || tfTipus.getText().isEmpty() ||
   tfRendszam.getText().isEmpty() || tfTipus.getText().isEmpty() || cbAllapot.getSelectedIndex() == -1) {
    JOptionPane.showMessageDialog(null, "Kérem töltse ki az összes adatot!", "hiba a felvitelben!", JOptionPane.WARNING_MESSAGE);
} else {
```

Ha minden adat meg van adva, a bejelentkezéshez hasonlóan itt is preparedStatementtel történik az adatbázisba írás.

```
Connection con = new Db().getCon();
String sql = "INSERT INTO rental_cars"
            + "(car_brand, car_type, lic_plate, daily_cost, status)"
            + "VALUES (?, ?, ?, ?, ?);";
PreparedStatement ps = con.prepareStatement(sql);
ps.setString(1, tfMarka.getText());
ps.setString(2, tfTipus.getText());
ps.setString(3, tfRendszam.getText());
ps.setString(4, tfNapiAr.getText());
ps.setInt(5, cbAllapot.getSelectedIndex());
ps.execute();
MainWindow.tablazatBetolt();
try {
    } catch (SQLException e1) {
```

4.3.6 EditDeleteCar osztály

Az autók bizonyos adatai megváltoztatására, és az autók törlésére szolgáló osztály. Ez az ablak minden megkapja a módosítani kívánt autó id.-jét a Cars ablaktól, ezt ki is jelzi felül az ablak nevénél, valamint betölti az autók adatait. Ezekből az adatokból kizártlag a rendszám és a napi ár módosítható, a márka, a típus, és az elérhetőség nem. Az elérhetőséget az autó kibérlésével és visszahozatalával tudjuk állítani.

Elemei:

```
private JPanel cpAutoModosit;
private JTextField tfRendszam;
private JTextField tfTipus;
private JTextField tfMarka;
private JTextField tfNapiAr;
@SuppressWarnings("rawtypes")
private JComboBox cbAllapot;
private JButton btnMentes;
private JButton btnKilepes;
private JButton btnTorles;
private JLabel lblUjAuto;
private JLabel lblMarka;
private JLabel lblTipus;
private JLabel lblRendszam;
private JLabel lblNapiAr;
private JLabel lblAllapot;
private Statement stm = null;
private Connection con = new Db().getCon();
```

Metódusai:

```
public void MentesGombraKattintas(Cars MainWindow, int carId) {..}

public void torlesGombraKattintas(Cars MainWindow, int carId) {..}

public void autoAdatokBetoltIdAlapjan(int carId) {..}
```

MentesGombraKattintas(cars MainWindow, int carId): Ellenőrzi hogy minden beviteli mező ki van e töltve, ellenkező esetben hibaüzenetet dob. Az ellenőrzés a következőképp zajlik:

```
if(tfMarka.getText().isEmpty() || tfTipus.getText().isEmpty() ||
   tfRendszam.getText().isEmpty() || tfTipus.getText().isEmpty() || cbAllapot.getSelectedIndex() == -1) {
    JOptionPane.showMessageDialog(null, "Kérlek töltse ki az összes adatot!", "hiba a felvitelben!", JOptionPane.WARNING_MESSAGE);
} else {
```

Ha minden szükséges adat meg van adva végez egy UPDATE-et az adatbázison az adott autón a beviteli mezők értékeit behelyettesítve.

```
String sql = "UPDATE rental_cars SET car_brand='"+tfMarka.getText()+"',"
           + "car_type = '"+tfTipus.getText()+"',"
           + "lic_plate = '"+tfRendszam.getText()+"',"
           + "daily_cost = '"+tfNapiAr.getText()+"',"
           + "status = '"+cbAllapot.getSelectedIndex()+"' WHERE car_id = '"+carId+"';";
```

A program ezután értesít minket a sikeres feltöltésről, újra tolta a Cars ablak táblázatát, és becsukja a módosító/törlő ablakot.

a **BtnMentes** gombra kattintva hívjuk meg.

Hibás felvitelnél hibaüzenetet kapunk.

torlesGombraKattintas(cars MainWindow, int carId):_a törlésre kattintva egy felugró ablak megkérdezi hogy biztos törölni szeretnénk e az autót, ezután ha

igen a válasz, id. alapján törli az autót az adatbázisból, ezután frissíti a Cars táblázatát, és bezárja a módosító/törlő ablakot.

A **BtnTorles** gombra kattintva hívhatjuk meg.

```
int valasz = JOptionPane.showConfirmDialog(null, "Biztosan törölni akarja az autót az adatbázisból?", "Biztos a dolgában?", JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.ERROR_MESSAGE);
String sql = "DELETE FROM rental_cars WHERE car_id = '"+carId+"'";
if(valasz==0) {
try {
stm.executeUpdate(sql);
MainWindow.tablazatBetolt();
JOptionPane.showMessageDialog(null , "Autó sikeresen törlve az adatbázisból", "Táblázat frissítve", JOptionPane.INFORMATION_MESSAGE);
dispose();|}
```

autoAdatokBetoltIdAlapján(int carId): ez a metódus az ablak indulásakor lefut, nem csinál mászt mint a kapott id. alapján lekérdezi annak az egy autónak az adatait, és beállítja őket a beviteli mezőkbe.

4.3.7 Customers osztály

A customers osztály szolgál az ügyfelek nyilvántartására, amire egy táblázat szolgál segítségül. Innen mehetünk tovább ha új ügyfeleket akarunk felvenni, illetve meglévők adatait akarjuk módosítani.

Szerepel még itt egy keresés mező is, aminek segítségével az ügyfelek nevére tudunk szűrni.

Elemei:

```
private JPanel cpUgyfelek;
private JScrollPane spUgyfelek;
private JTable tblUgyfelek;
private JTextField tfKeres;
private JButton btnKilepes;
private JLabel lblNev;
private JButton btnKeres;
private JButton btnUjUgyfel;
private JButton btnOsszesUgyfel;
```

Metódusai:

```
public void ujUgyfelAblakMegjelenit() {|
public void tablazatBetolt() {|
public void tablazatSorokTorol() {|
public void ugyfelekModositAblakMegjelenit(int tableRowIndex) {|
public void nevAlapjanKeres() {|

```

ujUgyfelAblakMegjelenit(): Ez a metódus létrehoz egy új példányt a NewCustomer osztályból.

tablazatBetolt(): Hasonlóan a **Cars** ablakhoz, lekrédezi az összes elemet a **rental_users** táblából, és sorba rendezi őket **user_id** alapján. A táblázat feltöltése előtt kitörli az összes elemet belőle a **tablazatSorokTorol()** metódussal, majd feltölti az újonnan lekérdezett adatokkal.

```
String sql = "SELECT * FROM rental_users ORDER BY user_id;"

while (rs.next()) {
    ((DefaultTableModel)tblUgyfelek.getModel()).addRow(
        new Object[] {
            rs.getString("user_id"),
            rs.getString("user_name"),
            rs.getString("user_address"),
            rs.getString("user_phone"),
        });
}
```

A metódus a frissítések kijelzésénél is fontos szerepet játszik, például egy ügyfél adatainak frissítése után is mindig meghívásra kerül, így a mentés után egyből látszódnak a már frissített adatok a táblázatban.

tablazatSorokTorol(): Ugyanúgy mint a **Cars** osztálynál, itt is végigmegy visszafele a táblázaton, és kitörli az összes sort.

```
int sorokSzama = tblUgyfelek.getModel().getRowCount();
for (int i = sorokSzama-1; i >= 0; i--) {
    ((DefaultTableModel)tblUgyfelek.getModel()).removeRow(i);
}
```

ugyfelekModositAblakMegjelenit(int rowIndex): A **Cars** osztályhoz hasonlóan itt is a táblázaton egy ügyfélre kattintva jeleníthetjük meg az **EditDeleteCustomer** ablakot ennek a metódusnak a segítségével. A **tableRowIndex** alapján megkeresi az ügyfél id.-ját a táblázatban, majd ezt tovább is adja az **EditDeleteCustomer** ablaknak.

nevAlapjanKeres(): Ez a kereső funkció a **btnKeres** gombra kattintva hívható meg. Értelemszerűen az ügyfelek név alapján való szűrésére szolgál, működik név részletekkel is, azt ellenőrzi hogy a kereső mezőbe beírt szövegrész megtalálható-e bármelyik ügyfél **user_name** mezőjében.

Keresésnél meghívódik a **tablazatSorokTorol()** metódus, majd feltölti a **tblUgyfelek** táblázatot a lekérdezett adatokkal.

Az SQL utasítás a következőképpen néz ki:

```
String sql = "SELECT * FROM rental_users WHERE user_name LIKE '%"+tfKeres.getText()+"%'";
```

4.3.8 NewCustomer osztály

A **NewCustomer** osztály szolgál az új ügyfelek adatainak rögzítésére az adatbázisba.

Elemei:

```
private JPanel cpUjUgyfel;
private JTextField tfNev;
private JTextField tfCim;
private JTextField tfTelefonSzam;
private JLabel lblUjUgyfel;
private JLabel lblNev ;
private JLabel lblCim;
private JLabel lblTelefonszam;
private JButton btnMentes;
private JButton btnKilepes;
```

Metódusai:

```
private void MentesGombraKattintas(Customers MainWindow) {
```

MentesGombraKattintas(Customers MainWindow): a **btnMentes** gombra kattintva hívhatjuk meg ezt a metódust, ami elmenti az új ügyfél adatait a `rental_users` táblában.

Gyakorlatilag nem csinál más, mint lefuttat egy `preparedStatement`-el megoldott **INSERT** sql utasítást, majd eltűnik, és mivel megkapta paraméternek a **Customers** MainWindow-t, így hozzáfér annak metódusaihoz is, így lefrissítheti az ügyfelek táblázatát a **MainWindow.tablazatBetolt()** parancsal.

```
private void MentesGombraKattintas(Customers MainWindow) {
    try {
        Connection con = new Db().getCon();
        String sql = "INSERT INTO rental_users"
            + "(user_name, user_address, user_phone)"
            + "VALUES (?, ?, ?);";
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString(1, tfNev.getText());
        ps.setString(2, tfCim.getText());
        ps.setString(3, tfTelefonSzam.getText());
        ps.execute();
        ps.close();
        JOptionPane.showMessageDialog(null , "Sikeres hozzáadás", "Táblázat frissítve", JOptionPane.INFORMATION_MESSAGE);
        dispose();
        MainWindow.tablazatBetolt();
    } catch (SQLException e1) {
        e1.printStackTrace();
    }
}
```

4.3.9 EditDeleteCustomer osztály

A meglévő ügyfelek adatainak módosítására, vagy az ügyfelek adatbázisból való törlésére szolgáló ablak. Meghívásakor megkapja a módosítani kívánt ügyfél id-jét, majd ezzel végez adatbázis műveleteket. A **Customers** ablakon egy ügyfélre kattintva jeleníthető meg, a beviteli mezőket automatikusan feltölti az ügyfelek adataival.

Elemei:

```
private JPanel cpUgyfMod;
private JTextField tfNev;
private JTextField tfCim;
private JTextField tfTelefonSzam;
private JLabel lblUgyfMod;
private JLabel lblNev;
private JLabel lblCim;
private JLabel lblTelefonSzam;
private JButton btnMentes;
private JButton btnTorles;
private JButton btnKilepes;
private Statement stm = null;
private Connection con = new Db().getCon();
```

Metódusai:

```
public void MentesGombraKattintas(Customers MainWindow, int userId) {}

public void ugyfelTorlesGombraKattintas(Customers MainWindow, int userId) {}

public void ugyfelAdatokBetoltIdAlapjan(int userId) {}
```

MentesGombraKattintas(Customers MainWindow, int userId):

a **btnMentes** gombra kattintva hívható meg, először ellenőrzi hogy minden beviteli mező ki van-e töltve, majd ha igen, frissíti (**UPDATE**) az ügyfél adatait a **rental_customers** táblában. Ellenkező esetben hibaüzenetet dob.

```
if(tfNev.getText().isEmpty() || tfCim.getText().isEmpty() || tfTelefonSzam.getText().isEmpty()) {
    JOptionPane.showMessageDialog(null, "Kérem töltse ki az összes adatot!", "hiba a felvitelben!", JOptionPane.WARNING_MESSAGE);
} else {
```

Az **update** itt is preparedStatement segítségével lett megoldva. Ha sikeres a frissítés az ablak bezárul, és a **Customers** ablak táblája frissítésre kerül. Ellenkező esetben hibaüzenet értesít minket a hibáról.

```
try {
    Connection con = new Db().getCon();
    String sql = "UPDATE rental_users SET user_name= ? ,"
        + "user_address = ? ,"
        + "user_phone = ? WHERE user_id = '" +userId+"'";
    PreparedStatement ps = con.prepareStatement(sql);
    ps.setString(1, tfNev.getText());
    ps.setString(2, tfCim.getText());
    ps.setString(3, tfTelefonSzam.getText());
    ps.executeUpdate();
    ps.close();
    JOptionPane.showMessageDialog(null , "Sikeres frissítés", "Táblázat frissítve", JOptionPane.INFORMATION_MESSAGE);
    MainWindow.tablazatBetolt();
    dispose();
} catch (SQLException e1) {
    JOptionPane.showMessageDialog(null , "Sikertelen hozzáadás", "Hiba", JOptionPane.ERROR_MESSAGE);
}
```

ugyfelTorlesGombraKattintas(Customers MainWindow, int userId):

A **btnTorles** gombra kattintva hívható meg ez a metódus, mely először felugró ablakban megkérdezi, hogy biztos törölni akarjuk e az ügyfelet, majd ha igen, egy egyszerű sql **DELETE** utasítással kitörli azt az egy ügyfelet az **user_id** alapján.

```
public void ugyfelTorlesGombraKattintas(Customers MainWindow, int userId) {
    int valasz = JOptionPane.showConfirmDialog(null, "Biztosan törölni akarja az Ügyfelet az adatbázisból?", "Biztos a dolgában?", JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.ERROR_MESSAGE);
    String sql = "DELETE FROM rental_users WHERE user_id = '" +userId+"'";
    if(valasz==0) {
        try {
            stm.executeUpdate(sql);
            MainWindow.tablazatBetolt();
            JOptionPane.showMessageDialog(null , "Az Ügyfél sikeresen törölve az adatbázisból", "Táblázat frissítve", JOptionPane.INFORMATION_MESSAGE);
            dispose();
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null , "Hiba!", "Hiba", JOptionPane.ERROR_MESSAGE);
        }
    } else if(valasz==1) {
        dispose();
    }
}
```

4.3.10 CarRent osztály

Az autók bérlesének adminisztrálására szolgáló osztály. Helyet kap rajta két táblázat, az egyik az épp szabad, a másik az éppen kibérelt autók megjelenítésére.

Ez az egyik leg összetettebb osztály, ez az elemei és metódusai számán is meglátszik

Elemei:

```
private JPanel contentPane;
private JTable tbSzabad;
private JTable tbFoglalt;
private JTextField tfRendszam;
private JTextField tfMarka;
private JTextField tfTipus;
private JTextField tfNapiAr;
private JScrollPane spFoglalt;
private JScrollPane spSzabad;
private JLabel lblSzabad;
private JLabel lblFoglaltAutk;
private JLabel lblAutFoglals;
private JLabel lblRendszm;
private JLabel lblMrka;
private JLabel lblTpus;
@SuppressWarnings("rawtypes")
private JComboBox cbUgyfel;
private JLabel lblUgyfel;
private JLabel lblBrlsKezdete;
private JLabel lblBrlsVge;
private JDateChooser dcKezdet;
private JDateChooser dcVge;
private JLabel lblAr;
private JButton btnLentes;
private JButton btnModosit;
private JButton btnAlaphelyzet;
private JButton button;
private JButton button_1;
private Statement stm = null;
private Connection con = new Db().getCon();
private java.util.Date RentDat, ReturnDat;
private java.sql.Date SqlRentDat, SqlReturnDat;
private JTextField tfRentId;
private JTextField tfSzabadId;
private JButton btnSzamitas;
private JTextField tfAr;
```

Metódusai:

```

public void tablazatokBetolt() {}

public void szabadTablazatSorokTorol() {}

public void foglaltTablazatSorokTorol() {}

public void autoAdatokTextfieldbeIras(int tableRowIndex) {}

public void foglaltAutoAdatokTextfieldbeIras(int tableRowIndex) {}

public void getCustomer() {}

public void MentesGombraKattintas() {}

public void alaphelyzetbeAllitas() {}

public void autoStatusUpdate() {}

private void maiDatumBeallitas() {}

private void szamitasGombraKattintas() {}

private void printGombraKattintas() {}

```

tablazatokBetolt(): A **Cars** ablakban lévő betöltéshez hasonlóan ez a metódus is a táblázatok feltöltésére szolgál, annyi különbséggel hogy itt a metóduson belül két sql utasítás fut le, két külön és ResultSettel tér vissza, amik külön-külön töltik fel a táblázatokat. Ennek a metódusnak a lefuttatásához van szükség a két táblázat sorainak törlését végző metódusokra. Ahogy eddig is láthattuk, először kitörli a már meglévő sorokat, és feltölti a táblázatokat az új adatokkal.

Az sql utasítások a következőképpen néznek ki:

```

String sql = "SELECT * FROM rental_cars WHERE status = 1 ORDER BY car_id;";
String sql2 = "SELECT * FROM rental_rents ORDER BY rent_id;";

```

A szabad autókat a **rental_cars** táblából kérdezzük le, szűrési feltételeként a status = 1 van beállítva, ami a szabad autókat jelöli, a **tbSzabad** táblázatba jelenítjük meg az adataikat.

A foglalt autók a **rental_rents** táblából lesznek lekérdezve és a **tbFoglalt** táblázatban jelenítjük meg.

szabadTablazatSorokTorol():

A **tbSzabad** táblázat sorainak törlésére szolgáló metódus.

foglaltTablazatSorokTorol():

A **tbFoglalt** táblázat sorainak törlésére szolgáló metódus.

autoAdatokTextfieldbelras(int tableRowIndex):

Új bérlet felvételéhez a szabad autók táblában rá kell kattintanunk egy autóra, és az adatai automatikusan kiíródnak a beviteli mezőkbe. Ezen mezők tartalma nem módosítható, ez csak információs céllal kerül kiírásra. Ez a metódus végzi az adatok beviteli mezőkbe írását.

Az autóra kattintva a `tableRowIndex` felhasználva a metódus kiszedi a táblázatból az autó id.-jét, és ezt felhasználva végez egy lekérdezést.

A kapott `resultSet` adataival töltjük fel a beviteli mezőket.

getCustomer():

Ez a metódus felel a `cbUgyfelek` comboBox feltöltéséért. Lekérdezi a `rental_customers` tábla összes elemét, és feltölti a comboBox-ot a rendszerben lévő emberek neveivel. Ezekből kell kiválasztanunk az ügyfelet aki ki szeretné bérlni az autót.

```
stm = con.createStatement();
String sql = "SELECT * FROM rental_users;";
ResultSet rs = stm.executeQuery(sql);
while(rs.next()) {
    String ugyfel = rs.getString("user_name");
    cbUgyfel.addItem(ugyfel);
}
```

mentesGombraKattintas():

Ez a metódus a `btnMentes` gomb megnyomásával hívható meg.

Több ellenőrzést is végez. Az első hogy ki van-e választva autó, van-e valami a textFieldekben? Ezután azt nézi meg hogy meg van-e adva az autó visszahozásának dátuma, majd hogy ki lett-e számolva a teljes ár, ezek nélkül minden eshetőségnél hibaüzenetet kapunk.

Ha kiválasztottunk egy autót, megadtuk mikor hozza vissza az ügyfél, és kiszámítottuk az árat, akkor menthetjük le a foglalást a `rental_rents` táblába.

Lefutásakor meghívásra kerül a `tablazatokBetolt()` és az `autoStatusUpdate()` metódus is.

Ez a mentés is `PreparedStatement`-el történik, a következőképpen néz ki a gyakorlatban:

```

Connection con = new Db().getCon();
String sql = "INSERT INTO rental_rents "
    + "(car_id, user_name, car_brand, car_type, lic_plate, rent_date, return_date, fee)"
    + "VALUES (?, ?, ?, ?, ?, ?, ?, ?);";
PreparedStatement ps = con.prepareStatement(sql);
ps.setString(1, tfSzabadId.getText());
ps.setString(2, cbUgyfel.getSelectedItem().toString());
ps.setString(3, tfMarka.getText());
ps.setString(4, tfTipus.getText());
ps.setString(5, tfRendszam.getText());
ps.setDate(6, SqlRentDat);
ps.setDate(7, SqlReturnDat);
ps.setInt(8, Integer.valueOf(tfAr.getText()));
ps.executeUpdate();
ps.close();

```

alaphelyzetbeAllitas():

Ez a metódus a **btnAlaphelyzet** gombra kattintva hívható meg, egyedül annyit csinál hogy az összes **textField** és **dateChooser** értékét alaphelyzetre állítja.

autoStatusUpdate():

Ez a metódus a **mentesGombraKattintas()** metódus futásakor kerül meghívásra, a szerepe pedig hogy amikor kibérelnek egy autót, annak státuszát frissíti szabadról foglaltra. Futása után frissíti a táblázatokat.

Ez az UPDATE is preparedStatement-el lett megoldva, a következőképpen néz ki:

```

public void autoStatusUpdate() {
    try {
        String carId = tfSzabadId.getText();
        Boolean status = false;
        String sql = "UPDATE rental_cars SET "
            + "status=?"
            + " WHERE car_id = '"+carId+"'";
        PreparedStatement ps;
        ps = con.prepareStatement(sql);
        ps.setBoolean(1, status);
        ps.executeUpdate();
        ps.close();
        tablazatokBetolt();
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Sikertelen hozzáadás", "Hiba", JOptionPane.ERROR_MESSAGE);
    }
}

```

maiDatumBeallitas():

Ez a metódus a **dcKezdet** dateChooser értékének megadására szolgál. Mivel a program úgy lett megtervezve, hogy minden napról induljon a bérlet, így fontos volt beállítani, hogy ez a mező is minden nap a napi dátumot jelezze ki, és így a bérletek az adatbázisba is minden nap úgy fognak felkerülni, amelyik napon kibéreltek őket.

Autóra kattintásnál, a textFieldek feltöltésével egyidőben van meghívva.

szamitasGombraKattintas():

A **btnSzamitas** gombra kattintva hívjuk meg ezt a metódust. Először ez is ellenőrzést végez, hogy biztosan kiválasztottunk-e egy autót, illetve megadtuk e a visszahozatal dátumát.

Ha megvannak ezek az adatok, a metódus először kiszámolja a napok különbséget, azaz, hogy hány napra bérlik ki az autót.

Ezután a napok számát felszorozza az autó napi árával, amiből adódik a bérbeadás teljes ára. Ezt az árat jeleníti meg nekünk a **tfAr** textField-ben.

Fontos, hogy az ár kiszámítása nélkül nem menthetünk el új bérlest, szóval ez egy elengedhetetlen lépés minden rögzítésnél.

```
if(tfNapiAr.getText().isEmpty() || tfMarka.getText().isEmpty() || tfTipus.getText().isEmpty() || tfRendszam.getText().isEmpty()) {  
    JOptionPane.showMessageDialog(null , "Kérem válasszon ki egy autót!", "Hiba", JOptionPane.ERROR_MESSAGE);  
} else if (dcVege.getDate()==null){  
    JOptionPane.showMessageDialog(null , "Kérem válassza ki a visszahozatal dátumát!", "Hiba", JOptionPane.ERROR_MESSAGE);  
} else {  
    long diff = dcVege.getDate().getTime() - dcKezdet.getDate().getTime();  
    long napok = TimeUnit.DAYS.convert(diff, TimeUnit.MILLISECONDS);  
    int napiAr = Integer.valueOf((tfNapiAr.getText()));  
    int teljesAr = (int) (napiAr*napok);  
    tfAr.setText(String.valueOf(teljesAr));  
}
```

printGombraKattintas():

A **btnNyomtatas** gombra kattintva kerül meghívásra. Feldob egy Windows nyomtatás ablakot, és kinyomtathatjuk vele az éppen foglalt autók listáját (**tfFoglalt**).

4.3.11 CarReturn osztály

A **CarReturn** osztály látja el azt a funkciót, ami egy bérlet letelte után rögzíti a visszahozott autókat a rendszerbe.

Elemei:

```
private JPanel contentPane;  
private JTextField tfRendszam;  
private JTextField tfMarka;  
private JTextField tfTipus;  
private JTextField tfUgyfel;  
private JTextField tfKeses;  
private JTextField tfElAr;  
private JTextField tfNapiAr;  
private JTable tbFoglalt;  
private JLabel lblAutoVissza;  
private JLabel lblRendszam;  
private JLabel lblMarka;  
private JLabel lblTipus;  
private JLabel lblUgyfel;  
private JLabel lblVissza;  
private JLabel lblKeses;  
private JLabel lblKibreitAukt;  
private DateChooser dcVisszaDatum;  
private JLabel lblAr;  
private JScrollPane spKiberelt;  
private JLabel lblBentLVAukt;  
private JScrollPane spSzabad;  
private JTable tbSzabad;  
private JButton btnNyomtats;  
private JButton btnVisszavetel;  
private int ar = 0;  
private Statement stm = null;  
private Connection con = new Db().getCon();  
private java.util.Date ReturnDat;  
private java.sql.Date SqlReturnDat;  
private JTextField tfszabadId;  
private JTextField tfFoglaltId;  
private int rentId;  
private JTextField tfTeljesAr;
```

Metódusai:

```
private void tablazatokBetolt() {}  
  
public void szabadTablazatSorokTorol() {}  
  
public void foglaltTablazatSorokTorol() {}  
  
private void kiszamitasGombraKattintas() {}  
  
public void foglaltAutoAdatokTextfieldbeIras(int tableRowIndex) {}  
  
public void visszavetGombraKattintas() {}  
  
public void autoStatusUpdate() {}  
  
private void foglaltTablabolTorles() {}  
  
private void printGombraKattintas() {}  
  
private void napiArMeghataroz() {}
```

tablazatokBetolt():

Mint a többi osztálynál is láthattuk, ez a metódus itt is a táblázatok adatainak betöltésére szolgál. Ebben az osztályban a **tbSzabad** és **tbFoglalt** táblázatokat tölti fel, itt is két sql utasítás és két ResultSet-et használunk. A táblázatok sorainak törlésére itt van szükség, azok a metódusok kizárolag itt kerülnek meghívásra.

Az sql utasítások:

```
String sql = "SELECT * FROM rental_returns;"  
String sql2 = "SELECT * FROM rental_rents ORDER BY rent_id;"
```

szabadTablazatSorokTorol():

Törli **tbSzabad** táblázat összes sorát, a **tablazatokBetolt()** metódusban alkalmazzuk.

foglaltTablazatSorokTorol():

Törli **tbFoglalt** táblázat összes sorát, a **tablazatokBetolt()** metódusban alkalmazzuk.

kiszamitasGombraKattintas():

A Metódus először ellenőrzéseket végez, először is, hogy van-e kiválasztott autó, utána pedig hogy meg lett-e adva a visszahozatal dátuma, valamint meg lett-e adva hány napot késsett az ügyfél. Ezek hiányában különböző hibaüzeneteket kapunk.

Az alkalmaztnak kell megadnia, hogy hány napot késsett az autó visszahozásával, majd ha megadta a **btnKiszamitas** gomb megnyomásával a metódus kiszámolja a felárat, amit kijelez a **tfFelAr** textField-ben.

foglaltAutoAdatokTextfieldbelras(int tableRowIndex):

A kibérelt autók adatainak beviteli mezőkbe írásáért felelős metódus. Meghívásához nem kell mászt tenni, mint rákattintani a **tbFoglalt** táblázatban egy autóra, és az adatait egyből megjeleníti a beviteli mezőkben.

A metódusban meghívásra kerül a **napiArMeghataroz()** metódus is, ez az esetleges felár meghatározásánál lesz fontos.

visszavetGombraKattintas():

Gyakorlatilag megfelel egy mentés gomb funkciójának, a **btnVisszavetel** gomb megnyomására kerül meghívásra.

Először ő is ellenőrzéseket végez, első sorban hogy van-e adat a beviteli mezőkben, azaz van-e kiválasztott autó, ha nincs akkor hibaüzenetet dob. Ezután a visszavétel dátumát ellenőrzi, amennyiben üres, hibaüzenettel tér vissza.

A metódus figyelembe veszi hogy késsett e az ügyfél az autóval vagy nem, ezért is kell kiszámolni a bérlet felárát. Amennyiben időben visszahozta az autót az előzetesen kiszámított ár kerül be végső árként, de ha késsett vele a felár ehhez hozzá adódik.

Amennyiben minden adat meg volt adva, preparedStatement segítségével végez egy beszúrást a **rental_rents** táblába, és rögzíti az autó visszahozását, frissíti az autó státuszát szabadra, a foglalt táblából eltávolítja, és frissíti az ablak táblázatait.

```
String sql = "INSERT INTO rental_returns "
+ "( lic_plate,user_name,return_date,fine)"
+ "VALUES (?,?,?,?);";
PreparedStatement ps = con.prepareStatement(sql);
ps.setString(1, tfRendszam.getText());
ps.setString(2, tfUgyfel.getText());
ps.setDate(3, SqlReturnDat);
ps.setInt(4, Integer.valueOf(tfFelAr.getText())+ar);
ps.executeUpdate();
ps.close();
autoStatusUpdate();
foglaltTablabolTorles();
tablazatokBetolt();
 JOptionPane.showMessageDialog(null, "Az autó visszahozva került!");

```

autoStatusUpdate():

Autók visszahozásánál frissíti a **rental_cars** táblában az adott autó státuszát szabadra, preparedStatement segítségével, majd frissíti a táblázatokat

```

String carId = tfFoglaltId.getText();
Boolean status = true;
String sql = "UPDATE rental_cars SET "
    + " status=?"
    + " WHERE car_id = '"+carId+"'";
PreparedStatement ps;
ps = con.prepareStatement(sql);
ps.setBoolean(1, status);
ps.executeUpdate();
ps.close();
tablazatokBetolt();

```

foglaltTablabolTorles():

Autó visszahozásánál törölni kell a **rental_rents** táblából az autót, erre nyújt megoldást ez a metódus.

```

try {
    String sql = "DELETE FROM rental_rents WHERE rent_id =" + rentId;
    Statement stm = con.createStatement();
    stm.executeUpdate(sql);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(null, "Hiba!", "Hiba", JOptionPane.ERROR_MESSAGE);
}

```

printGombraKattintas():

A **tbSzabad** – szabad autók táblázatának kinyomtatására szolgáló metódus, a **btnNyomtatas** gombra kattintva hívható meg.

```

try {
    tbSzabad.print();
} catch (Exception e2) {
    JOptionPane.showMessageDialog(null, "Sikertelen nyomtatás!", "Hiba", JOptionPane.ERROR_MESSAGE);
}

```

napiArMeghataroz():

A Napi ár meghatározására szolgáló metódus, id. alapján végez egy lekérdezést a **rental_cars** táblában, és egy rejtett **tfNapiAr** textField-be írja bele a napi árat, ami felhasználható lesz utána későbbi számításokra.

```

int carId = Integer.parseInt(tfFoglaltId.getText());
try {
    stm = con.createStatement();
    String sql = "SELECT * FROM rental_cars WHERE car_id = '" + carId + "'";
    ResultSet rs = stm.executeQuery(sql);
    if(rs.next()) {
        tfNapiAr.setText(rs.getString("daily_cost"));
    }
} catch (SQLException e) {
    JOptionPane.showMessageDialog(null, "Hiba!", "Hiba", JOptionPane.ERROR_MESSAGE);
}

```

5 TESZTELÉS

Az alábbi példák alapján bemutatnák pár különböző esetet a program hibakezeléseiről, a program felhasználói interakciókra történő történő reakcióiról.

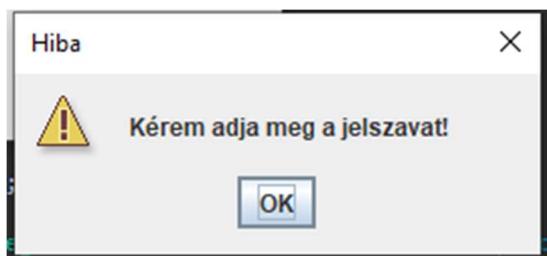
5.1 BEJELENTKEZÉS

A program indítása után, már a bejelentkező oldalon is találkozhatunk egy hibakezeléssel, ami megakadályozza illetéktelen személyek belépését a program fő felületeire.

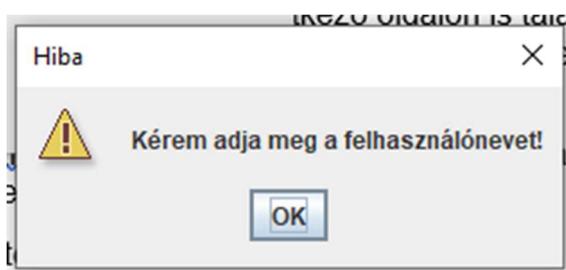
Ez először megnézi, hogy minden bejelentkezási adat ki van-e töltve, amennyiben nem, ezekről felugró ablakban értesít minket.

Amennyiben meg vannak adva a bejelentkezási adatok, a program összeveti a megadott bejelentkezási adatokat az adatbázisban található adatokkal, és csak akkor enged tovább, ha helyes adatokat adtunk meg.

Hiányzó jelszó esetén a következő ablak jelenik meg:



Míg hiányzó felhasználónév esetén a következő:



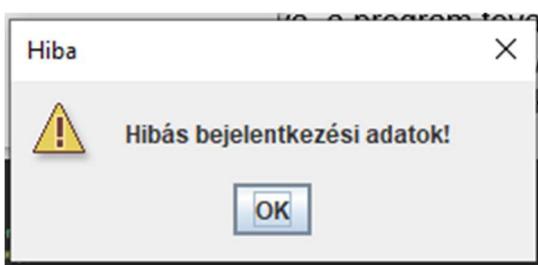
Ez az ellenőrzés a **carsAblakMegjelenít** metódus elején található, és kód szinten a következőképpen van megoldva:

```
if(tfUserName.getText().isEmpty()) {  
    JOptionPane.showMessageDialog(null, "Kérém adja meg a felhasználónevet!", "Hiba", JOptionPane.WARNING_MESSAGE);  
} else if (password.isEmpty()){  
    JOptionPane.showMessageDialog(null, "Kérém adja meg a jelszavat!", "Hiba", JOptionPane.WARNING_MESSAGE);  
} else {
```

Amennyiben az adatok meg vannak adva, a program tovább halad, és végez egy preparedStatement-es sql lekérdezést. Amennyiben van ilyen rekord az adatbázisban eltűnik a bejelentkező oldal, és a **Cars** ablakra irányít minket a program.

```
else {
    PreparedStatement ps;
    try {
        ps = con.prepareStatement(sql);
        ps.setString(1, userName);
        ps.setString(2, password);
        ResultSet rs = ps.executeQuery();
        if(rs.next()) {
            frmLogin.setVisible(false);
            dispose();
            new Cars();
        } else {
            JOptionPane.showMessageDialog(null, "Hibás bejelentkezási adatok!", "Hiba", JOptionPane.WARNING_MESSAGE);
            tfUserName.setText("");
            pfPwd.setText("");
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "SQL hiba!", "Hiba", JOptionPane.WARNING_MESSAGE);
    }
}
```

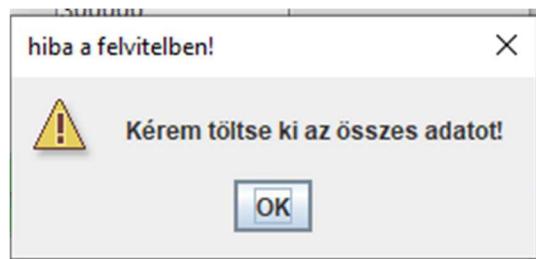
Amennyiben helytelen adatokat adunk meg, egy hibaüzenet értesít minket erről, és a beviteli mezőket alaphelyzetbe állítja.



5.2 AUTÓ MÓDISÍTÁSA / TÖRLÉSE

Autó módosításakor az **EditDeleteCar** ablakon a **btnMentes** gombra kattintva egy **JOptionPane** felugró ablak értesít minket a módosítás sikerességről.

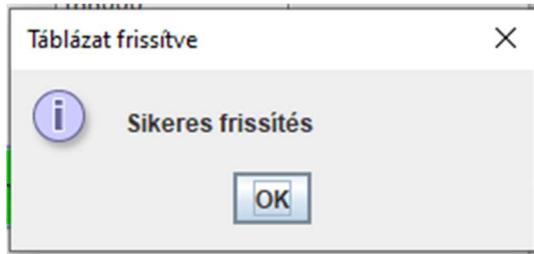
Ezelőtt a program még végez egy ellenőrzést, mely a **MentesGombraKattintas** metódusban a legelső lefutó kódrészlet, amiben megnézi, hogy minden adatot megadtak-e az autóhoz. Amennyiben nincs meg minden adat a program hibaüzenettel értesít minket erről.



Ez a kódban a következőképpen van megoldva:

```
if(tfMarka.getText().isEmpty() || tfTipus.getText().isEmpty() ||
   tfRendszam.getText().isEmpty() || tfTipus.getText().isEmpty() || cbAllapot.getSelectedIndex() == -1) {
    JOptionPane.showMessageDialog(null, "Kérem töltse ki az összes adatot!", "hiba a felvitelben!", JOptionPane.WARNING_MESSAGE);
} else {
```

Amennyiben viszont megadtunk minden adatot a metódus elvégzi az adatbázis frissítést, amelynek sikerességéről egy felugró ablak értesít minket.



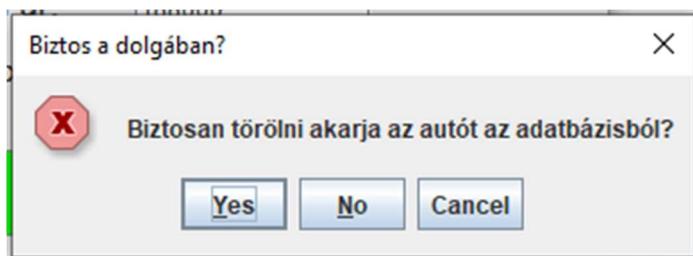
A kódban ez a következőképpen néz ki:

```
JOptionPane.showMessageDialog(null, "Sikeres frissítés", "Táblázat frissítve", JOptionPane.INFORMATION_MESSAGE);
```

Autó törlésekor a **btnTorles** gombra kattintva itt is egy **JOptionPane** jelenik meg, viszont itt először ki kell választanunk hogy biztosan törölni szeretnénk e az autót. Erre a **JOptionPane.YES_NO_CANCEL option** attribútuma ad lehetőséget. Ha az igen gombra kattintunk, a program lefuttatja az SQL utasítást ami kitöri az autót az adatbázisból, ellenkező esetben egyszerűen bezárja az ablakot. Ez a kódban a következőképpen van megoldva:

```
int valasz = JOptionPane.showConfirmDialog(null, "Biztosan törölni akarja az autót az adatbázisból?", "Biztos a dolgában?",
                                             JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.ERROR_MESSAGE);
String sql = "DELETE FROM rental_cars WHERE car_id = '"+carId+"'";
if(valasz==0) {
try {
    stm.executeUpdate(sql);
    MainWindow.tablazatBetolt();
    JOptionPane.showMessageDialog(null, "Autó sikeresen törölve az adatbázisból", "Táblázat frissítve", JOptionPane.INFORMATION_MESSAGE);
    dispose();
} catch (SQLException e) {
    JOptionPane.showMessageDialog(null, "SQL hiba", "Hiba", JOptionPane.ERROR_MESSAGE);
}
} else if(valasz==1) {
    dispose();}
```

A felugró ablak pedig így néz ki:



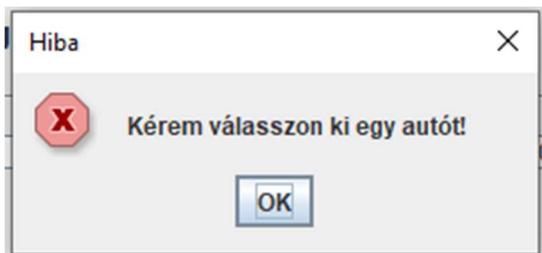
Hasonló megoldással találkozhatunk az ügyfelek módosítására és törlésére szolgáló ablakon (**EditDeleteCustomer**).

5.3 AUTÓ BÉRLÉSNÉL FELLÉPŐ HIBAKEZELÉSEK

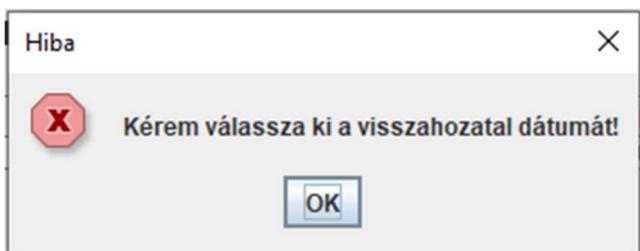
Mivel a **CarRent** osztályom összetettebb műveleteket végez, így igyekeztem itt is minden hibakezelést felhasználóbarátan és érthetően megoldani. Itt is minden hibáról hibaüzenet értesít minket, és végigkísér az esetleges hibalehetőségeken.

Alaphelyzetben sem a **btnMentes**, sem a **btnSzamitas** gomb nem használható, mivel a mentés előfeltétele az ár kiszámítása, ennek előfeltétele pedig egy kiválasztott autó, valamint az időintervallum kiválasztása amíg bérlni szeretnék az autót. minden hiányzó esetben felugró ablakok jelzik a hiányosságokat

Kiválasztott autó híján a **btnSzamitas** gombra kattintva:



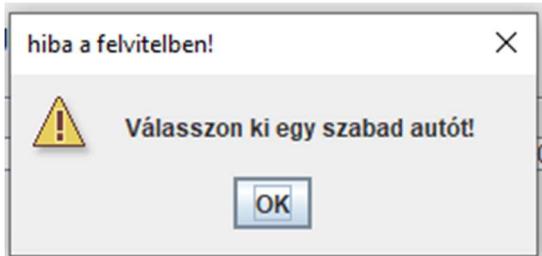
Kiválasztott autóval, de a befejezés dátuma nélkül:



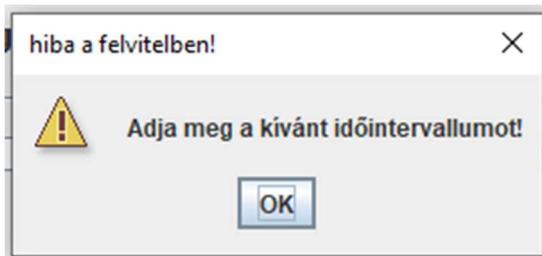
Ezek a következőképpen lettek megoldva a **szamitasGombraKattintas** metódusban:

```
if(tfNapiAr.getText().isEmpty() || tfMarka.getText().isEmpty() || tfTipus.getText().isEmpty() || tfRendszam.getText().isEmpty()) {
    JOptionPane.showMessageDialog(null, "Kérem válasszon ki egy autót!", "Hiba", JOptionPane.ERROR_MESSAGE);
} else if (dcVege.getDate()==null){
    JOptionPane.showMessageDialog(null, "Kérem válassza ki a visszahozatal dátumát!", "Hiba", JOptionPane.ERROR_MESSAGE);
} else {
    // Folytatás a többi kód részére
}
```

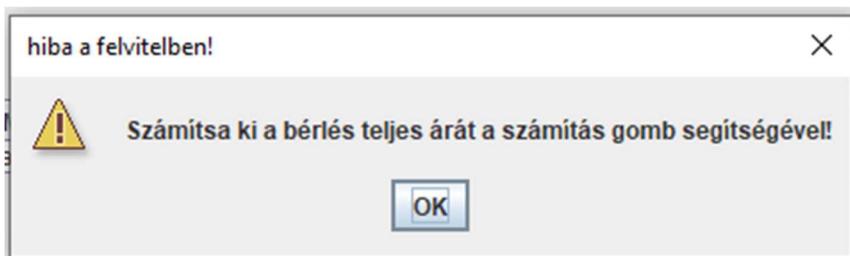
A **mentesGombraKattintas** metódusban pedig 3db hibakezelés oldja meg az esetleges hiányosságokat. Az első, ha nincs kiválasztva autó akkor a következő hibaüzenet jelenik meg:



Ha van kiválasztott autó, de a befejezés dátuma hiányzik:



Ha pedig kiválasztottuk a visszahozatal dátumát, de nem számítottuk ki az árat a **btnSzamitas** gombra kattintva, akkor a következő üzenet jelenik meg:



Ezek a hibakezelések a kódban a következőképpen lettek megoldva a **mentesGombraKattintas** metódus elején:

```
if(tfMarka.getText().isEmpty() || tfTipus.getText().isEmpty() || tfRendszam.getText().isEmpty()) {
    JOptionPane.showMessageDialog(null, "Válasszon ki egy szabad autót!", "hiba a felvitelben!", JOptionPane.WARNING_MESSAGE);
} else if(dcKezdet.getDate()==null || dcVege.getDate()==null) {
    JOptionPane.showMessageDialog(null, "Adja meg a kívánt időintervallumot!", "hiba a felvitelben!", JOptionPane.WARNING_MESSAGE);
} else if(tfAr.getText().isEmpty()) {
    JOptionPane.showMessageDialog(null, "Számítsa ki a bérlet teljes árát a számítás gomb segítségével!", "hiba a felvitelben!",
        JOptionPane.WARNING_MESSAGE);
} else {
    try {
```

6 ÖSSZEFoglalás

6.1 SZAKDOLGOZATOM CÉLJA

A szakdolgozatom célja egy olyan összetett asztali alkalmazás elkészítése volt, aminek a valóéletben is lehet némi haszna, és akár egy cég számára is megfelelő megoldást nyújton. Úgy gondolom a program használatát egyszerű lehet

elsajátítani akár egy informatikában nem jártas személynek is, így egy cég bármely alkalmazottjának alkalmas lehet a használatra.

Ezen felül úgy gondolom az eddig megszerzett tudásom hasznosításában, az önálló felhasználásának a gyakorlásában is nagyon hasznos volt ez a projekt, sok dolgot kellett felelevenítenem a készítése során, ami mindenkorban pozitív dolog volt számomra.

6.2 MEGVALÓSÍTÁS

Az első lépés az adatbázis létrehozása volt, ezt követte az ablakok kinézetének megtervezése, létrehozása. Fontos volt az egységesség, így igyekeztem egységes elnevezéseket használni a textFieldek, gombok, és egyéb elemek elnevezésénél.

Az ablakok elkészítését követően kezdődött az összetettebb munkafolyamat, a metódusok megírása. Először a **Db** osztály készült el az adatbázis kapcsolat megteremtésére, majd innen haladtam sorban, először a **Login** bejelentkező oldalt csináltam meg, utána a **Cars** következett, majd az ehhez tartozó **NewCar** és **EditDeleteCar**.

Miután az autók részével készvoltam, az ügyfelek következtek. Itt is hasonló módon készítettem el az oldalakat, először a **Customers**, utána a **NewCustomer**, majd az **EditDeleteCustomer** oldalt. Majd mikor ezekkel kész voltam, a **CarRent** és a **CarReturn** oldalakat. Őszintén szólva ezek bizonyultak a leg összetettebb résznek, ezek programozásával töltöttem talán a legtöbb időt a projekt során.

6.3 FEJLESZTÉSI LEHETŐSÉGEK

- Annak megoldása, hogy a visszavét oldalon ne kelljen manuálisan megadni a késés napjainak számát, hanem a visszahozás dátumából és az előre meghatározott dátumból számolja ki.
- Az új alkalmazottak hozzáadására szolgáló felület létrehozása.
- A program telepítő fájljának létrehozása, amellyel telepíthetjük a végső verziót bármely számítógépre.

6.4 ÖSSZEGZÉS

Összességében nagyon hasznosnak éreztem ennek a projektnek az elkészítését, voltak dolgok, amik eleinte akadályt jelentettek, de ezek megoldása hasznos tapasztalatokkal járt és ezek megoldását is végül sikeresként élhettem meg, amikor végre összeállt a program.

6.5 IRODALOMJEGYZÉK

6.5.1 Internetes irodalomjegyzék

<https://stackoverflow.com/>

<https://www.javatpoint.com/>

<https://docs.oracle.com/>

<https://www.tutorialspoint.com/>

<https://dzone.com/>

6.5.2 Nyomtatott irodalomjegyzék

[Robert C. Martin - Tiszta kód](#)

[Besedin Andrei – Amazing Java](#)